Simulating Uninhabited Combat Aircraft in Hostile Environments (Part II)

Philippe Morignot  
Jean-Clair Poncet  
Johan Baltié  
AXLOG Ingénierie  
19-21, rue du 8 mai 1945  
ARCUEIL, France  
+33 (0)1 41 24 31 19, +33 (0)1 41 24 31 34, +33 (0)1 41 24 31 22  
philippe.morignot@axlog.fr, jean-clair.poncet@axlog.fr, johan.baltie@axlog.fr

Patrick Fabiani  
Eric Bensana  
Jean-Loup Farges  
ONERA-CERT / DCSD / CD  
BP 4025  
31055 TOULOUSE, France  
+33 (0)5 62 25 27 83, +33 (0)5 62 25 29 01, +33 (0)5 62 25 27 76  
fabiani@cert.fr, bensana@cert.fr, farges@cert.fr, tessier@cert.fr

Bruno Patin  
Dassault Aviation  
78, quai Marcel Dassault  
92552 SAINT-CLOUD Cedex, France  
+33 (0)1 47 11 58 54  
bruno.patin@dassault-aviation.fr

Keywords:  
Vehicle system modeling, aerospace simulation, autonomous agents

ABSTRACT: We present a simulation of a team of humanly-piloted combat aircraft which coexist with uninhabited ones (UCAVs) in missions consisting in attacking targets in hostile territories. Last year [14], we presented a first simulation involving two aircraft (an aggressive one and a conservative one) in a realistic but simple scenario involving attacking a target with one unexpected pop-up threat. This year, we present the software architecture of each aircraft and a practical / theoretical analysis to enable limited on-board computation and distributed reasoning among the 8 aircraft. This project is no more in its design phase and is now in its coding phase.

1. Introduction

Uninhabited aircraft control has been much explored during the last few years [3] [15], meaning the development of combat aircraft capable of autonomy on a spectrum from total guidance from the ground (“drones”), to full autonomous flight (goals, flight plans and flight laws are entirely set by the aircraft itself). In addition, we consider a package of 2 to 8 aircraft, instead of a unique aircraft. This reach better realism but entails more complex reasoning on the aircraft role inside the package, and even leads to considering splitting a package into 2 or more sub-packages: one sub-package deciding to attack a pop-up threat, while the other goes around the radar range, a synchronization point being previously decided to form again the initial package after this intermediate attack.

1 This work is funded by Délégation Générale pour l’Armement (the French DARPA) through the contract ARTEMIS. The authors thank Nelly Strady-Lécubin, Catherine Tessier, Jean-François Gabard, Stéphane Millet and Pierre Hélie for insightful contributions to this work.
The main difficulty of the missions considered in this paper is that usually things do not occur as planned, because, for example, the location of foe radars are not entirely known, if any, in advance by the package aircraft: although a first path and plan can be computed before the mission starts, it is very unlikely to stay valid for the whole mission. Hence, re-planning at execution time (i.e., during the flight) is necessary to adapt to unexpected changes in the environment (e.g., a pop up threat). Similarly, a device of one aircraft of the package may break down, leading the whole package to re-organize itself in response, as an attempt at carrying on its mission despite local failures.

If human pilots are heavily trained for adequately responding to such unexpected events, reproducing such reasoning inside computer programs embedded in UCAVs remains difficult: the complexity of the algorithms involved is exponential (formally, the problems involved are NP-hard), which inevitably leads to very long computations in the worst case (even on the fastest computers), thus leading to a response which may simply be obsolete when finally delivered by the algorithm. Therefore combining long deliberation with fast reaction in order to obtain an intelligent behavior of the UCAVs is the major theoretical topic of our study.

In this paper, we focus on the simulation of the aircraft (system, reactive and decisional functions) and its environment. A future project will involve porting the developed source code on 3 real small aircraft, to check for real the feasibility of the approach developed so far. After a first project, dedicated to the same problem but with relaxed constraints, has been successfully carried out [16], the reported project adds reasoning in limited time (i.e., the reasoning part of the aircraft must not take longer to deliberate than a fixed amount of time) and in a distributed way (i.e., no leader aircraft computes the plan of the other aircraft, all aircraft reason on the same hierarchical level, and must negotiate to reach a consensus). In addition, the whole simulation must be homothetic to real-time, i.e., no time step must last much longer than the others, because of algorithms of exponential complexity to launch during it. As another justification, the future development of the speed of computers will lead to progressively reduce the real time taken by a time step, progressively leading the simulation to full real time.

The paper is organized as follows: first, we briefly sum up the domain of targets’ attack by autonomous combat aircraft; then we present the software architecture used to represent the various software components of each aircraft; then, we present a mathematical model solving the time constraint and distributed reasoning constraint inside the planner software component; then we describe implementation details; finally, we sum up our contribution and present our future research directions.

2. Domain

2.1. UCAVs attacking targets

The mission of a package of uninhabited aircraft is to attack targets in hostile territories (see Figure 1).

The aerial space is divided into flight corridors, one package per corridor --- in order to avoid collision among different packages involved in different missions. Corridors can be considered as 4D volumes (e.g., cylinders), inside which no other friendly package is guaranteed to fly.

Several prioritized targets (less than 10) can be set for attack before the mission starts.

At the aircraft level, each aircraft owns a limited number of resources, e.g., fuel, weapons, chaff (i.e., electromagnetic reflectors for countermeasures), flares (i.e., infrared reflectors also for countermeasures); it also owns devices that are likely to break down during the mission, e.g. an intra formation data link (IFDL, a little-detectable link), a low bandwidth data link (LBDL, used to communicate at medium distance with the command & control base and an easily detectable link), a GPS device, a radar, a thruster; it also owns unlimited resources, e.g. an electromagnetic jammer or a radar warning receiver. Carrying out a mission successfully amounts to maximizing the probability of survival ($p_s$) and the probability of killing ($p_k$, probability of destroying the targets) given the other constraints on the aircraft i.e. resource availability and aircraft maneuverability. The $Ps$ table is given as a function of...
the kind of threat and its distance to the aircraft; the \( Pk \)
table is given as a function of the kind of the target and
the available weapons; hence \( Ps \) and \( Pk \) computations
may be performed anyway along the path. The fuel
consumption is given by a table as a function of the
speed and altitude. Limiting the consumption of
resources also results in avoiding making sharp turns or
sharp altitude changes (and it is the same at the
package level): consequently, a path (i.e. a sequence of
corridors) preventing sharp maneuvers has to be chosen
in advance among the set of the possible corridors.

At the package level, a major trajectory constraint is
related to collision avoidance, resulting in minimum
and maximum distances between any pair of aircraft in
the package. This also results in changes of the
package shape during turns or altitude changes, e.g. the
aircraft at the far left side of the package may fly to the
right side of the package after a turn, to limit fuel
consumption. Another constraint is to coordinate the
resource consumption among the package members:
for example, if one aircraft happens to detect the firing
of a ground-to-air missile, a subset of the package
(possibly not including the initial aircraft) may drop
flares, because they have more flares available than the
initial aircraft. Similarly, if the package is detected by a
ground radar, the sub-package which is formed to
destroy this intermediate target is composed of the
aircraft owning enough weapons to perform the attack:
the main goal being to destroy the main target, this
entails that enough weapons must always be kept on
board at least one aircraft of the package, despite
intermediate targets on the way.

During the mission, the aircraft in the package
communicate intensely within the friendly territory or
in the attack phase to warn each other (or the control &
command base) about the next trajectory they will
adopt, or about the time when they will drop bombs.
They also communicate after an intermediate attack to
reach the meeting point (where both sub-packages
merge again into a single one). A device of one aircraft
may break down during the mission: if this happens to
the package leader, another leader with better devices
is elected; if this happens to a wingman, \( Ps \) and \( Pk \) are
re-evaluated and the package is reconfigured as far as
possible to give a less important role to this wingman
(and to increase \( ps \) and \( pk \) again). If a vital device
breaks down, the mission may be aborted, which
results in some or all the aircraft returning to the base.

2.2. Assumptions

Several simplifying assumptions are made:

- The attack of the targets itself is not represented. When reaching targets (one node

of the graph of corridors and waypoints), one
or several aircraft drop bombs, and the
complex trajectories of the aircraft during the
attack phase is not taken into account.

- No details on take off or landing.
- No air-to-air combat. Only air-to-ground (e.g.,
an aircraft dropping bombs on a target) and
ground-to-air (e.g., a foe radar lightning an
aircraft) fights are considered.

Now the problem is to organize the various kinds of
knowledge involved in the problem: domain, decisional
and reactive knowledge. The purpose of the next
section is to present one way for performing this.

3. Software Architecture

In this section, we present the way the software are
organized inside one aircraft (see Figure 2).

The software simulating the aircraft are separated into
the system functions of the aircraft (e.g., flight
management system, autopilot, package maintenance
system, device for launching flares, device for
communicating) and the control of these equipments
(i.e., the mission management system, or MMS). Strictly
speaking, the MMS receives status on the
equipment of the aircraft (sensors) and sends
commands to them (effectors).

![Figure 2: Software architecture of the Mission
Management System of each aircraft.](image)

35 years ago, the architectures of agents were linear:
perception, then deliberation, and then action [4].
Although intuitive, the drawback of this architecture is
that there might be a very long time between
perception and action, due to the exponential complexity of the algorithms involved in deliberation — as a result, the agent may get stuck reasoning, while the environment changes and requires attention.

To our knowledge, there is no theory indicating the agent architecture that everyone should adopt. But more recent architectures (e.g., [1]) organize an agent’s architecture as composed of 2 or 3 levels, with at least one level for algorithms of exponential complexity (decisional level) and one level for algorithms of polynomial complexity (reactive level). As a result, the agent can keep perceiving/acting while deliberating and reasoning — two separate processes, one per level, which communicate. This improves the survival of the agent in fast changing environments: In case of emergency, when a decision has to be taken, agents with such architectures can engage in fast reactive loops (reactive level), while deliberation occurs (decisional level) and tries to come up with a neat solution as a response to a pressuring environment. We choose such a two-level architecture for the aircraft’s MMS (see Figure 2).

The components of the reactive level include:

- **Observe & Compare**: This component computes the immediate probability of survive (Ps) and of kill (Pk), and computes safety areas around the aircraft (spheres around the aircraft, altitude) to avoid inter-aircraft or ground collision. It compares these values to thresholds and, if checked, passes the information on to the Preempt component for immediate safety reaction. In nominal mode (i.e., if these thresholds are not exceeded), information is passed on to the Act component, for normal unfolding of the current plan.

- **Preempt**: This component must react in emergency situation, i.e., in case of danger (see safety information passed by the previous component). It selects and launches a contingency plan, which is a sequence of actions (e.g., pulling up if the aircraft is too close to the ground, making a sharp turn if another aircraft is too close, launching flares and chaff if a foe missile has been detected, adopting an avoidance trajectory if a pop up threat has been detected). The appropriate contingency plan is passed on to the Act component for merging to the current plan and for execution.

- **Act**: This component receives the current plan to execute from the Plan component or the Preempt component, unstacks the first action from the plan, accordingly sends commands to the system functions of the aircraft, and checks that the action has been correctly executed. Action execution can be attached to an instant, to a location or to a criterion (e.g., in parallel with another action, immediately after another action, after some time after another action). It is responsible for merging a contingency plan with the current plan, e.g., if an aircraft loses its communication (no response from this other aircraft for some time), the aircraft changes its altitude to reach a predefined altitude slot, one different slot per aircraft, in order to avoid colliding other aircraft.

The components of the decisional level include:

- **Long-Term Observe**: This component computes the long term probabilities of survive (Ps) and of kill (Pk). As opposed to the computation performed in the Observe & Compare component, these probabilities are integrated on the planned trajectory of the aircraft (planned Ps and Pk) — this requires that the current plan be provided by the Plan component.

- **Compare**: Similarly to the Compare part of the Observe & Compare component, this component compares the long term Ps and Pk to thresholds, and, if exceeded, decides to re-plan, i.e., find a new path and its attached actions.

- **Prepare**: This component, purely technical, formats data passed on to the Plan component.

- **Plan**: This component computes a new path, and its attached actions, given the current situation of the aircraft (see section 4).

- **Format**: This component, purely technical, formats the plan before passing it on to the Act component.

The Plan components of different aircraft can directly communicate (without passing information through the other reactive and decisional components), to enable a distributed reasoning (see section 4.4.).

The connection among these components (see Figure 2) follows two modes:
In the **nominal** mode, the Observe & Compare component gathers data from the system functions of the aircraft, checks that no threshold is exceeded, and passes the current status on to the Act component. This latter component receives the status information (e.g., completion of the current action), and keeps following the plan, which is declared as still valid.

In the **emergency** mode, the Observe & Compare component notices a low Ps or Pk, activates the Preempt component for choosing a contingency plan, and at the same time passes information on to the Long-Term Observe component, for computing the long term Ps and Pk (which will be below the threshold, since their immediate values are below it too), so the Compare component will decide to re-plan. The Act component executes the contingency plan provided by the Preempt component, and waits for the new plan to come from the Plan component.

Typically, the aircraft detects a foe radar lightning it (emergency mode), launches an evasive maneuver (a trajectory avoiding the radar detection range) while the planner computes a new plan for gathering the aircraft of the package, eventually after a sub-package has decided to attack this radar.

The behavior resulting from this software architecture models the human pilot’s one (intelligently reacting to unexpected events from the environment). But as said above, the difficulty lays in the long computation time taken by the Plan component --- exponential complexity of the algorithms involved in it. Therefore, in the next section we focus on that component, to explain how we can add constraints to it, by imposing computation time limit and distributing the reasoning it performs.

### 4. (Re-)Planning

#### 4.1. Context

Each aircraft must find (i) a path through corridors from their take off to their landing (i.e., a sequence of waypoints), and (ii) a sequence of actions to take on this path (e.g., drop bombs, launch flares, activate counter-measures, illuminate a target with a laser). Each item leads to different techniques: item (i) leads to designing a path-planner, either discrete (a path in a graph) or continuous (a 3D trajectory); while item (ii) leads to designing an action planner, in the sense of a logical unfolding of actions described in some language, which constitute a plan proving that some goals will hold in some final situation, given an initial situation (from [4] to the last planners in Artificial Intelligence, e.g., see the ICAPS conference series). Fortunately, pilots mainly search for a discrete path, and then attach actions to time windows on arcs of this path, with the guarantee that there exists an instant (or a duration, if the action lasts) at which the action will be able to be executed. Therefore, path-planning dominates action planning, and there is no need to embed an action planner in the sense of [4] --- path planning is only needed.

The variables of the problem are hierarchically decomposed --- a heuristics on the order in which the concepts are handled:

- **Goals**: they are made of prioritized targets, with the least priority goal being a return to base without bombing.
- **Threats**: the current threats are identified.
- **Assignment**: Assigning UCAVs to threats and goals.
- **Synchronization points** (e.g., after a split into 2 sub-packages, one attacking a threat and the other flying around the radar detection range).
- **4D path**.

Re-considering one choice performed at one level impacts all the level below it: backtracking occurs in the implicit heuristics drawn by this hierarchy. For example, if no path is found by the path planner (5th level), new synchronization points are set (4th level). Similarly, if a new threat appears (2nd level), this may either lead to backtrack to the goals (1st level, for example, leading the package to return to the base) or re-launching the 3rd, 4th and 5th levels for taking this new threat into account. Hence, control flows back and forth through the hierarchy of these levels.

Technically, the 1st level (goals) is performed by a set of rules. The 2nd level (threats) is also performed by a set of rules, with global constraints such as the total number of resources needed to handle a threat (either attack or ignore an intermediate threat, given the total number of bombs available and the number of them which must be dropped on the main target). The 3rd level (assignment) is also performed by a set of rules (e.g., if one UCAV has disappeared, replacing it by a similar one to perform the desired task). The 4th level (synchronization) is performed by reasoning on constraints, e.g., when an event occurs, domain knowledge is used to decide whether to split/rejoin at a synchronization point or not (if a split/rejoin action is
decided, the location of these actions is determined here).

The 5th level (path planning) is performed at two hierarchical (sub-)levels: in an upper level, a discrete path is searched for in the graph of corridors and waypoints. Then, at a lower level, between two successive waypoints, the corridor is discretized into a smaller graph, leading to a second path search. If the upper level has a fixed complexity, the lower level can be set to an arbitrary complexity, depending on the step chosen for this discretization.

Once waypoints have been determined (in the main graph, and then in its sub-graphs), the actual continuous trajectory (a potential 6th level) of each aircraft is interpolated by using its flight laws, given its flight capabilities (included into the system functions of the aircraft, at the lower part of Figure 2).

4.2. Model

Formally, the corridors and waypoints are modeled as a graph \((C, W)\), where \(W\) are the nodes (waypoints) and \(C\) are the arcs (corridors). The take off, landing and targets points are modeled as nodes through which the package must fly. There are approximately 80 waypoints \(W\) in the missions we consider. A similar graph is built for sub-waypoints inside one corridor. The dimension of this sub-graph depends on the desired detail level. The 5th-level problem is to find a path inside this graph (and sub-graphs) of possible paths.

The 3rd dimension is represented by adding an altitude variable to each waypoint (its domain is composed of the altitude slots). This slightly increases the combinatorial complexity, but on the other hand this increase is limited by the flight capabilities of the aircraft --- its possible trajectories regarding altitude.

Let us describe the model used for path planning with synchronization and aircraft grouping. Let \(i\) be a node in \(W\), \(I_{in}\) be the incoming arcs of node \(i\) and \(O\), \(O_{out}\) its outgoing arcs. Let \(u_{ij} = 1\) iff the package uses the arc that goes from waypoint \(i\) to waypoint \(j\), and 0 otherwise. Then, following [7], the following equation (1) holds (Kirchoff law, or more generally flow conservation law):

\[
\forall i \in W, \sum_{j \in I_{in}} u_{ij} - \sum_{k \in O_{out}} u_{ik} \leq 1 \quad (1)
\]

The first term represents the number of incoming arcs used by the package to arrive at waypoint \(i\). The second term is similar, but for arcs outgoing from \(i\). Equation (1) merely expresses that the number of arcs used by the package at waypoint \(i\) is at most \(1\) --- it can be exactly set to 1 for the take off and landing waypoints (with the second/first term only, respectively) and for the targets’ waypoints.

Now we have to model the various consumptions and actions that an UCAV can take at a node/waypoint \(i\). Let \(C\) be the fuel level available at node \(i\) in an UCAV. Then, equation 2 expresses the fuel consumption on arc \((i,j)\):

\[
\forall \text{arc}(i, j), u_{i,j} = 1 \Rightarrow C_j = C_i - c_{i,j} \quad (2)
\]

where the term \(c_{i,j}\) depends on the length or the arc \((i,j)\) and on the speed on the aircraft on this arc. Similar equations can be written for the time of arrival \(D\) of the aircraft on a node \(i\) (since waypoints are expressed as 4D points, instead of 3D), the variation of the probability of survive \(Ps\) between two successive nodes of an arc \((i, j)\), the quantity of bombs available at each node (depending on whether the node is a target or not), etc.

Now the preceding model is set for one agent (one package or one UCAV), whereas we want to model a package of UCAVs, and even sometimes several packages with synchronization points. One way to model this is to add an index \(k\) to each preceding variable and constant, meaning that it is valid for UCAV \(k\) only. For example, \(u_{i,j,k} = 1\) iff UCAV \(k\) flies on arc \((i,j)\); \(P_{s,k} = I\) iff UCAV \(k\) flies over node/waypoint \(i\) --- this last variable is equal to the first or second term of equation (1) and should not be confused with the probability of survive \(Ps\).

The grouping of aircraft into (sub-)packages is left open --- it will be determined by the collaboration needs (see section 4.4). The only equation linking UCAVs together relates to synchronization points: Let \(A_{i,k} = 1\) iff the node/waypoint \(i\) needs to be attacked (because it is an intermediate or final target) by UCAV \(k\), and 0 otherwise (e.g., if node \(i\) is not a target, then \(A_{i,k} = 0\) for all \(k\)). Let \(I\), the time at which target on node/waypoint \(i\) is attacked (i.e., an integer between 0 and the maximal duration of the mission). Then, equation (3) holds:

\[
\forall \text{node } i, \forall \text{aircraft } k, A_{i,k} = 1 \Rightarrow I = D_{i,k} \quad (3)
\]

That is, if node \(i\) has to be attacked by UCAV \(k\), then the arrival time of UCAV \(k\) at this node is equal to the time at which the target is attacked --- this implicitly
forces all involved UCAVs to arrive at the same time at node \( i \).

Finally, equations (1), (2) and (3) can be simplified for the extreme points in the implicit hyperspace. For example, if a node/waypoint is a target according to the 1st level, then at least one UCAV attacks it and equation (1) is an equality (instead of an inequality). Or, the fuel consumption \( C_i \) is set at the maximum value of its domain \( C_{\text{max}} \) at take off time. Or, if a node is a target, at least one UCAV must fly through this node and all participating UCAVs must be there at the same time. Or, the domain of each variable ranges from its minimum value (e.g., 0 for a \( P_s \)) to a maximum value (e.g., 1 for a \( P_s \) at the instant at which the package takes off at the take off node/waypoint, \( C_{\text{max}} \) for fuel consumption \( C_i \)), etc.

The following assumptions are made in the model (essentially to keep it linear, see section 5):

- The speed of an aircraft is constant on an arc \((i,j)\).
- The computation of the \( P_s \) variation on an arc takes into account only the number of threats threatening this arc.
- The fuel consumption is on an arc does not depends of speed, height and slope.

### 4.3. Reasoning under time constraint

As envisioned in [14], the main concept for reaching this constraint is to use anytime algorithms [20]: this kind of algorithms has the property of gracefully improving the quality of the solution as a function of the allocated computation time. One example of such algorithms is local optimization (e.g., tabu search [5]), i.e., iteratively searching in the neighborhood of an initial state and keeping the best neighborhood found so far in one variable. As a result, when a time out is exceeded, the best solution found so far is returned --- and its quality increases over time, hence the anytime property.

The used local search is decision & repair [8], a variation on constraint programming: the idea of the algorithm is to traverse the variable/value tree as performed in regular constraint programming, but, when an empty domain is detected by propagation (i.e., no solution), to look for a solution in neighborhoods of the reached point by relaxing constraints --- as opposed to backtracking to a previous choice point and choosing to explore another branch of the tree. As a result, the number of relaxed constraints in one neighborhood characterizes the quality of the solution exhibited by the algorithm.

### 4.4. Distributed reasoning

#### Communication groups

Distributing the previous time-constrained reasoning involves having the aircraft communicating to exchange data --- needed by the distributed algorithms involved (see next sub-section). But unfortunately, at war time, the aircraft cannot communicate all the time: for example, some modes of jamming prevent communication, even at short distance (IFDL); or, if one aircraft flies at low altitude on one side of a mountain and another aircraft flies on the other side of this mountain, electromagnetic waves cannot be broadcast through this mountain and reach the other aircraft. In other words, if communication bandwidth is already limited by IFDL and LBDL, communication at all is also limited.

We cope with this problem by introducing communication groups: aircraft which are members of the same communication group can freely communicate. The communication groups vary along time, allowing different configurations of aircraft in communication groups as the mission unfolds. This is a hard domain constraint on the possible distribution (see next section) of the previous model.

#### Algorithms for collaborative problem solving

To our knowledge, collaboration inside a communication group can be obtained using the following types of paradigms:

**Model decomposition (or feasible methods)** [5]. This type of methods is iterative. The agent at the top of the hierarchy sets the value of the interaction variables. Then, the agents at the bottom of the hierarchy solve their local problems respecting the value of the interaction variables. They also inform the agent at the top of the hierarchy of the sensitivity of the solution of the local problem to variations in interaction variables. Using this information the agent at the top of the hierarchy modifies the value of the interaction variables.

**Goal coordination (or dual-feasible methods)** [17]. This type of methods is also iterative. The agent at the top of the hierarchy set the criteria used by the agents at the bottom of the hierarchy. Then, the agents at the bottom of the hierarchy solve their local problem with full freedom for the interaction variables. The agent at the top of the hierarchy modifies the local criteria while the values of the interaction variables are not consistent with the constraints of the global model.
Successive approximations methods [12]. This type of method is also iterative. Each agent optimizes the global criterion with respect to its own variables. If an improvement is obtained by an agent, the current solution is updated.

Selection of propositions [2] [10]. This type of method is not iterative and involves basically two steps. Each agent at the bottom of the hierarchy formulates a set of propositions about the actions it could perform. The agent at the top of the hierarchy optimizes the global criterion and satisfies the interaction constraints by selecting a single proposition by agent. Optionally during a third step each agent at the bottom of the hierarchy refines the selected proposition.

Let us model a collaboration problem as a set of local problems composed of local constraints on local variables \( x_i \), and a set of (in)equality constraints \( g(x) = g(x_1, ..., x_n, ...) = 0 \) on the sum of functions \( g(x_i) \) on local variables, linking local problems together. A global function \( f(x) = f(x_1, ..., x_n, ...) \) is a sum of functions on local variables \( f(x_i) \) and has to be minimized --- it measures the overall quality of the solution obtained by collaboration.

The idea of collaboration by costs is that, without linking constraints, the local problems are independent, hence they can be solved separately. Therefore, this kind of collaboration will try to set these linking constraints to nil, in order to separate the resolution of the local problems.

One help comes from the minimax theorem [9]: if there exists a saddle point \( (x^*, \lambda^*) \) for the expression \( f(x) + \lambda g(x) \), i.e., \( x^* \) is a maximum of this expression, while \( \lambda^* \) is its minimum (hence the name of theorem), then \( x^* \) is the minimum of \( f(x) \) on every \( x \) such that \( g(x) = 0 \), and \( g(x^*) = 0 \). The idea is to identify \( f(x) \) to the function to minimize, and to identify \( g(x) \) to the linking constraints: if such a saddle point exists, then according to this theorem, the linking constraints disappear and \( f(x^*) \) is the minimum of the criterion, without the linking constraints --- this saddle point is a solution of the collaboration problem.

Therefore, goal coordination may be obtained by iteratively looking for this saddle point of the function \( f(x) + \lambda g(x) \). This can be done either by fixing \( \lambda \), considered as \( \lambda^* \), in which case the local problems, with the criteria \( f(x_i) + \lambda g(x_i) \), can independently be solved; or by fixing \( x \), considered as \( x^* \), in which case the dual variable \( \lambda \) can be adjusted to satisfy the linking constraints: if \( g(x) \) is positive (resp. negative), the saddle point will be looked for by augmenting (resp. decreasing) \( \lambda \); if \( g(x) = 0 \), then we have reached a saddle point, and by the way a solution of the collaboration problem. We choose this second option: An algorithm iteratively optimizes local problems and then increments or decrements \( \lambda \) in order to search for a saddle point.

Finally, note that the above minimization and maximization can obviously be reversed, by inserting a minus sign in the involved functions and constraints.

Model for collaboration (extracted from the model of section 4.2.)

Let us refine the model of the section 4.3. Let \( T_t = 1 \) iff target \( t \) is attacked, and 0 otherwise --- there is a mapping from the node indices to the target indices. Let \( P_{\text{surv}} \) be the sum of the probabilities of survive \( Ps \) of each UCAV \( k \) (characterizing the survivability of the mission as a whole). Let \( E_{\text{glob}} \) be the sum of the attack’s efficiency \( E_t \) on target \( t \), weighted by coefficient \( \omega_t \). Let each efficiency \( E_t \) on target \( t \) be the sum of the attack modes \( M_{t,m} \) (i.e., \( M_{t,m} = 1 \) iff target \( t \) is attacked with mode \( m \)) weighted by the efficiency \( e_{m} \) of each mode \( m \) (i.e., an integer between 0 and 100).

It should be noted first that, with this new variable on attack modes, a target is attacked iff one attack mode is set:

\[
\forall \text{target } t, \quad T_t = \sum_m M_{t,m} \tag{4}
\]

Then, an intuitive idea for the global function to be maximized (i.e., measuring the quality of a mission) is:

\[
\max_{t, P_{\text{surv}}, E_{\text{glob}}} \left( \alpha \sum T_t + \beta P_{\text{surv}} + E_{\text{glob}} \right)
\]

where \( \alpha \) and \( \beta \) are user-defined weights --- the global efficiency \( E_{\text{glob}} \) is not weighted, since it is a sum of already weighted terms \( E_t \) (see above). This function to be maximized expresses that the collaboration problem consists in maximizing the number of attacked targets, of the survival of the UCAVs and of the global efficiency. We now reformulate this expression, in order to reach elementary terms.

Firstly, by replacing \( T_t \) by its expression in equation (4), \( P_{\text{surv}} \) by the sum of individual \( Ps \), and \( E_{\text{glob}} \) by its expression as a weighted sum of \( E_t \), the function to maximize becomes:
\[
\max_{M_{t,m}, P_s} \left( \alpha \sum_t \sum_m M_{t,m} + \beta \sum_k P_s^k + \sum_i w_i E_i \right)
\]
given the additional constraint that there cannot be more than one attack mode per target (a consequence of equation (4)):

\[
\forall t \arg \text{ et } t, \sum_m M_{t,m} \leq 1 \tag{5}
\]

Secondly, by replacing \( E_i \) by its expression as a weighted sum of the attack modes \( M_{t,m} \), and by setting \( v_{t,m} = \alpha + w_i e_{m} \) to group the terms \( M_{t,m} \) together, the above function to be maximized becomes:

\[
\max_{M_{t,m}, P_s} \left( \sum_t \sum_m v_{t,m} M_{t,m} + \beta \sum_k P_s^k \right)
\]

Now that the function to maximize is as simple as possible, we can introduce the dual variables in it. Remember that for this, the constraints linking local problems together must be identified. They are, firstly, equation (3), which is reformulated here to exhibit a term equal to zero:

\[
\forall \text{ node } i, \forall \text{ aircraft } k, A_{i,k} (D_{i,k} - I_i) = 0 \tag{6}
\]

And, secondly, the equation expressing that a node \( i \), if attacked by UCAV \( k \) \((A_{i,k} = 1)\), must be flown over by an UCAV \( k \) \((P_{i,k} = 1)\). It can be reformulated in a similar way as:

\[
\forall \text{ node } i, \forall \text{ aircraft } k, A_{i,k} (P_{i,k} - 1) = 0 \tag{7}
\]

Finally, by introducing the first term of equation (6) with a set of dual variables \( \mu_{i,k} \) to be minimized (in order to get the left hand side of equation (6) as close to 0 as possible), and by similarly introducing the first term of equation (7) with a second set of dual variables \( \kappa_{i,k} \) to be minimized for the same reason, we come up with an expression to be both maximized and minimized (but on different variables):

\[
\max_{M_{i,m}, P_s} \min_{A_{i,k}, D_{i,k}, I_i, P_{i,k}, \mu_{i,k}, \kappa_{i,k}} \left( \sum_t \sum_m v_{t,m} M_{t,m} + \beta \sum_k P_s^k + \sum_i \mu_{i,k} A_{i,k} (D_{i,k} - I_i) + \sum_i \kappa_{i,k} A_{i,k} (P_{i,k} - 1) \right)
\]

This exactly is a function for which a saddle point (maximization of the variables \( M_{i,m}, P_s^k, A_{i,k}, D_{i,k}, I_i \) and \( P_{i,k} \); minimization of the dual variables \( \beta, \mu_{i,k} \) and \( \kappa_{i,k} \)) has to be searched for, with the heuristic method explained in the previous sub-section.

\section*{5. Implementation}

The system functions of the aircraft are implemented in ATHENA, a generic simulation environment based on the language C++ [19]. They will later be ported to a simulation environment supporting a High Level Architecture [6] --- one federate per aircraft, linked by a Run-Time Infrastructure for managing time unfolding.

The software architecture is developed in the language C++. Messages exchanged among the MMS components follow the XML format (header/body). The planner is located in a separate thread of the MMS Linux process, hence it is the only asynchronous component of the architecture.

The planner is developed in CHOCO [11], a constraint programming language developed in the language JAVA --- equations (1) to (5) are straightforwardly expressed into this language. Part of it has also been experimented in ILOG CPLEX, a simplex algorithm for mixed integer programming [13], taking advantage of the fact that equations (1) to (5) are linear (an implication such as equation (2) can be expressed as two linear inequalities). The distributed version of this algorithm is implemented in JAVA.

When compared to [14], the Petri net tool ProCoSa [18] has been discarded, despite its intrinsic interest, because (i) the logic of activation of the components of the software architecture can be expressed (see above) without a need for complex activation mechanisms, and (ii) all software components, except the planner, are synchronous (no need for ProCoSa’s external processes).

\section*{6. Conclusion and Future Work}

In this paper, we have presented a simulation of a package of uninhabited combat aircraft having to attack targets in hostile territories --- a follow up on [14]. After having presented the complexity of the actions involved in such missions, we have presented a 2-level software architecture of the MMS, and its components, for carrying out deliberation and reaction at the same time. More precisely, we focused on the main deliberative software components, the planner, and showed (i) a (linear) model for representing the problem of path planning under synchronization, threats and goals constraints, (ii) an algorithm that can reason within time constraints (by using a variation of constraint programming, decision and repair), and (iii)
the notion of communication group, and an algorithm for collaboration by costs and the associated reformulation of the previous model to fit into its frame. Finally, we have presented implementation choices.

The project currently is in its coding phase, hence we are not capable of showing results yet --- work in progress (first demonstrations are expected in June 2005).

As part of this work, a study on the relationship between embedded decisional software and norms/standards has also been carried out and will be published elsewhere.

Future work includes:

- Modeling the trajectories of the aircraft in the attack phase.
- Extensively testing the developed simulation on increasingly complex scenarios, and checking that we have reached homothetic real time and realistic aircraft behavior.
- Experimenting simulations with one or several aircraft controlled by a human pilot (interoperability) --- no MMS for these aircraft.
- Porting the whole simulation on 3 real aircraft (e.g., noisy sensors). An envisioned intermediate step could be to port only some aircraft on real ones, still leaving the others in simulation.

7. References

[18] Available at URL http://www.cert.fr/dcsd/cd/PROCOSA
Author Biographies

Dr. Philippe MORIGNOT is chief scientific officer at AXLOG Ingénierie, Arcueil, France. He supervises and participates in the Research & Development projects.

Jean-Clair Poncet is project manager at AXLOG Ingénierie, Arcueil, France. He supervises the AXLOG part of the Misure project and participates in the Artemis project.

Johan Baltié is software engineer at AXLOG Ingénierie, Arcueil, France. He participates in the Research & Development projects, including UAVs projects Misure and Artemis.

Dr. Patrick FABIANI is a research scientist in the Systems Control and Flight Dynamics Department at ONERA, Toulouse, France. He supervises and participates in autonomous UAV research projects.

Dr. Eric BENSANA is research scientist in the Systems Control and Flight Dynamics Department at ONERA, Toulouse, France. He currently works on problem solving algorithms for planning and scheduling and also on autonomy for space applications.

Dr. Jean-Loup FARGES FARGES is a research scientist in the Systems Control and Flight Dynamics Department at ONERA, Toulouse, France.

Bruno PATIN is a Research & Development engineer at Dassault Aviation, Saint-Cloud, France. He currently works on distributed simulation tools and is in charge of trying autonomy algorithms for UAVs.